

master

SelfStudyIRM / OS\_and\_Programming / main.c

ThomasCarstens format



1 contributor

Raw Blame



430 lines (357 sloc) 8.84 KB

```
1 #include <stdio.h> //printf
2 #include <stdlib.h> //malloc
3 #include <unistd.h> //open write read close
4 #include <string.h>
5
6 #include <sys/types.h>
7 #include <sys/stat.h>
8 #include <fcntl.h>
9 #include <pthread.h> //pour le threading (makefile a modifier)
10
11 #include <errno.h> //errno perro
12 //char buffr;
13 //char * fgets(char *restrict s, int n, FILE *restrict stream);
14
15 // SCHEDULER: pour creer une interface
16 #define PROCESS_THREAD int main() {
17 #define END_PROCESS }
18
19 // //DIFFERENCE STACK AND HEAP
20 // dans /proc/pid/ voir cat maps
21 #define STR_SIZE 30
22 #define ALLOCATE(SIZE) (malloc(SIZE))
23
24 int main(){
25     int a;
26     int *b = malloc(sizeof(int));
27     printf("%i %p %p\n", getpid(), &a, b);
28     scanf("%i", &a);
29     return 0;
30 }
```

```
31
32
33 // PYRAMID: MY WAY
34 int main() {
35
36     char branches=3;
37     char asterisk=atoi(str);
38     char bigasterisk=1;
39     char row = 0;
40     for (int row=0; row<atoi(str); row++){
41         asterisk--;
42
43         for (int i=0; i<asterisk; i++){
44             printf("-");
45         }
46         for (int i=0; i<bigasterisk; i++){
47             printf("*");
48         }
49         printf("\n");
50         bigasterisk=bigasterisk+2;
51     }
52
53 }
54
55 //WITH SCANF://///////////////
56
57 int pyramid() {
58
59     char branches=3;
60     char asterisk=atoi(str);
61     char bigasterisk=1;
62     char row = 0;
63     for (int row=0; row<atoi(str); row++){
64         asterisk--;
65
66         for (int i=0; i<asterisk; i++){
67             printf("-");
68         }
69         for (int i=0; i<bigasterisk; i++){
70             printf("*");
71         }
72         printf("\n");
73         bigasterisk=bigasterisk+2;
74     }
75
76     int read_integer(){
77         int read_number;
78         scanf ("%i", &read_number);
79         return read_number;
80     }
81 }
82
```

```

83  int main() {
84      int taille = read_integer();
85      pyramid(taille);
86      return 0;
87  }
88
89
90  //FGETS: GREGOR IMPLEMENTATION
91  // 1 lire une ligne de stdin
92  // a. Allouer la mémoire pour cette string
93  char *str = ALLOCATE(STR_SIZE * sizeof(char));
94  // b. Lecture au clavier
95  fgets(str, STR_SIZE, stdin);
96
97  // 2 l'afficher sur stdout
98  printf("%s\n", str);
99
100 return 0;
101 }
102
103 str buff[10];
104
105 int *a = malloc(30*sizeof(int));
106 //mettre les 30 int a 0
107 //a[1]= 0;
108 for (int i=0; i<30; i++){
109     *(a+i)=i;
110     //printf("%i\n", *a);
111     sprintf(buff,"%i\n", *a);
112     if i=30 {
113         a[i]='\0';
114     }
115 };
116 printf(*a);
117 free (a);
118 return 0;
119
120
121 //FGETS AVEC INSPECTION
122 char name[10]="a";
123 // printf("Who are you? ");
124 printf("%s", name);
125 while (atoi(name)!=1){
126     fgets(name,10,stdin);
127 };
128 printf("%s.n",name);
129 return(0);
130
131
132 //POINTERS
133 int a = 42;
134 int *b = &a;

```

```

135 printf("%p\n", &a);
136 return 0;
137 while(fgets(char buffr, 10 , stdin) != NULL)
138 {
139     printf("%s\n", buffr);
140 }
141
142 void f(int *a){
143     *a = 42;
144 }
145 // ecrire sur un pointeur
146 // ecrit au prealable moi meme
147 int main() {
148     int a;
149     f(&a);
150     printf("")
151 }
152
153 //ALLOUER MANUELLEMENT MEMOIRE ET VOIE CA DANS /proc/pid/ cat maps
154 int main() {
155     printf("%i %p\n", getpid(), sbrk(0));
156     int a;
157     scanf("%i", &a);
158     return 0;
159 }
160
161 //Write a program that asks for numbers (integers), stores them in the heap and when
162 // no number is supplied, prints the total sum of what was inputted.
163 //NOT VERIFIED.
164 int main(){
165
166     reps=5;
167     for (int i=0; i<reps; i++){
168         char *b = malloc(10 * sizeof(int));
169         fgets(b[i], 10, stdin);
170         printf("%i\n", *b[i]);
171         sum += *b[i];
172         printf(sum);
173     }
174
175 }
176
177 //STRUCTS
178 //NOT VERIFIED.
179 struct Point {
180     float x;
181     float y;
182 }
183
184 int main() {
185     struct Point a_point = {1.0, 1.0}; //dans ma stack
186     printf("%f", a_point.x);

```

```

187     struct Point *b_point= malloc(sizeof(struct Point)); //dans ma heap
188     printf("%f", b_point -> x);
189     //meme chose que *(b_point).x
190     return 0;
191 }
192
193
194 //////////////////////////////////////////////////
195 // //STRUCT TYPEDEF
196 typedef struct
197 {
198     int x;
199     int y;
200 } the_structure;
201
202 int main(){
203     the_structure a;
204     a.x = 0.0f;
205     return 0;
206 }
207
208
209
210 struct Point {
211     float x;
212     float y;
213 };
214
215 int main() {
216     char name[10]="a";
217
218     printf("vector components: \nx: ");
219     fgets(name, 10, stdin);
220     a_point.x = atoi(name);
221     printf("\ny: ");
222     fgets(name, 10, stdin);
223     a_point.y = atoi(name);
224     int centroid = (a_point.x+a_point.y)/2;
225     printf("\ncentroid: %i", centroid);
226     return 0;
227     //CORRECTION: FOR MULTIPLE POINTS
228     // find nb of points to input. i=nb_points
229     for (int i=0; i<q; i++){
230         float x, y;
231         scanf("%f %f", &x, &y);
232         struct Point my_point = {1.0, 1.0}; //dans ma stack
233         numbers[i] = my_point;
234         //then compute centroid
235         for (int z=0; z<q; z++){
236             sum_x += numbers[z].x;
237             sum_y += numbers[z].y;
238         }

```

```

239     }
240 }
241
242 //////////////////////////////////////////////////
243 char buf[128];
244 // //FILEMANIP with syscalls. Ca peut aussi
245 //fonctionner avec :strings vers l'internet"
246 // NOT VERIFIED
247 #define BUFFER_SIZE 1024
248 int main(int argc, char const *argv[]){ //argc=nb args
249                                     //argv -> char* -> item
250     char buffr[10]; //!= malloc(50*sizeof(char))
251
252     void *buffer = malloc(BUFFER_SIZE);
253     printf ("%s\n", argv[1]);
254     int fd = open (argv[1], O_RDONLY);
255
256     //int fd = open("testFile", O_CREAT, O_RDWR);
257     if (fd== -1) {
258         perror("open");
259         return EXIT_FAILURE;
260     };
261
262     char str[]= "Bonjour,";
263
264     while(1){
265         ssize_t read;
266         long bytes_read = read (fd, buffer, BUFFER_SIZE);
267         if (bytes_read== -1) {
268             perror("read");
269             return EXIT_FAILURE;
270         };
271
272         if (bytes_read==0){
273             break
274         }
275
276         long bytes_written = write(STDOUT_FILENO, buffer, bytes_read);//write(fd, str, 1);
277         if (bytes_written== -1) {
278             perror("write");
279             return EXIT_FAILURE;
280         };
281
282         printf("fd: %d\n", fd);
283     }
284     //printf("bits: %ld\n", read(fd, buffr, 50));
285     //printf("buf: %s\n", buffr);
286     close(fd);
287     free (buffr);
288 }
289
290 //////////////////////////////////////////////////

```

```

291 //FILEMANIP with FILE* abstraction.
292 //https://www.tutorialspoint.com/c_standard_library/c_function_fread.htm
293 int main(){
294     char buf[128];
295     FILE *file=fopen("testFile.txt", "w+");
296     //size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
297     //fread(void *ptr, size_t size, 50, *file);
298
299     char d[] ="Hello stream";
300     //fwrite(const void *ptr, size_t size, size_t nmemb,
301             //FILE *stream);
302     fwrite(d, strlen(d)+1, 1, file);
303     fseek(file, 0, SEEK_SET);
304     /* Read and display data */
305     fread(buf, strlen(d)+1, 1, file);
306     printf("%s\n", buf);
307
308     fclose(file);
309 }
310 //pointeur sur un tableau sur une chaine de caracteres
311
312
313 // //stat --format "%B" file;
314 // //OUTPUT SIZE OF MY FILE
315 //https://linuxhint.com/stat-system-call-linux/
316 int main(int argc, char const *argv[]){
317     struct stat my_stat;
318
319     if (stat(argv[1], &my_stat) == -1){
320         perror("stat");
321         return 1;
322     }
323     printf ("size: %ld\n", my_stat.st_size);
324     printf ("stat fichier: %ld\n", my_stat);
325     return 0;
326 }
327
328
329 ///////////////////////////////////////////////////
330 //THREADS
331 //GREGOR
332 void function(){
333     printf("bonjour\n");
334 }
335
336 int main (int argc, char const *argv[]){
337     //int (*f1_ptr) {long, int} = &f1;
338     //(*f1_ptr)
339     pthread_t thread1;
340     pthread_create(&thread1, (NULL), (void * (*)(*)void *)) &function, NULL);
341     pthread_join(thread1, NULL); //BLOQUANTE
342     //non teste, mais possible de trigger avec scanf("%i", thread1)

```

```

343     return 0;
344 }
345
346 // TXA IMPLEMENTATION
347 //makefile: gcc -pthread main.c -o main
348 // actually causes the linking
349 // to be done by the linker
350
351 //I just learned passing by ref only works in C++
352 //NOT FINISHED.
353 int counter;
354 pthread_mutex_t lock;
355
356 void* Thread1(int counter) {
357     printf("Hello world from other thread!\n");
358     pthread_mutex_lock(&lock);
359     counter++;
360     pthread_mutex_unlock(&lock);
361     //printf("%d", counter++);
362     int sleeptime =1;
363     //sleep(sleeptime);
364     return NULL;
365 }
366
367 void* Thread2(int counter) {
368     printf("Hello world from other thread!\n");
369
370     pthread_mutex_lock(&lock);
371     counter++; //critical section.
372     pthread_mutex_unlock(&lock);
373
374     printf("%d", counter);
375     int sleeptime =1;
376     //sleep(sleeptime);
377     return NULL;
378 }
379
380 int main() {
381
382     pthread_mutex_init(&lock,NULL);
383
384     pthread_t t1;
385     pthread_t t2;
386     printf("Spawning thread.\n");
387     pthread_create(&t1, &counter, (void*)Thread1, NULL);
388     pthread_create(&t2, &counter, (void*)Thread2, NULL);
389
390     for (int i = 0; i < 100; ++i)
391     {
392         if (pthread_join(t1, NULL) != 0)
393         {
394             fprintf(stderr, "error: Cannot join thread # %d\n", i);

```

```
395     }
396 }
397 for (int i = 0; i < 100; ++i)
398 {
399     if (pthread_join(t2, NULL) != 0)
400     {
401         fprintf(stderr, "error: Cannot join thread # %d\n", i);
402     }
403     pthread_mutex_destroy(&lock);
404 }
405
406 printf("Program end.\n");
407 return 0;
408 }
409
410
411 //////////////////////////////////////////////////
412 // ERROR DETECTION
413 // int main(int argc, char const *argv[]){
414 //     char* str = malloc(50*sizeof(char));
415 //     if (str!=NULL){
416 //         // il y a eu erreur
417 //         perror("malloc");
418 //         return 1; //fonction indique donc l'erreur
419 //         //return EXIT_FAILURE; //alternatif
420 //     }
421 //     return 0;
422 // }
423 //apt-get install moreutils
424 //errno -1 //POUR VOIR LES ERREURES PROGRAMME
425
426
427 // int main(){
428 //     printf("O_CREAT %i\n", O_CREAT);
429 //     open("fichier", O_CREAT | O_SYNC); //| veut dire ou bit a bit
430 // }
```